Homework #7

[ECE30021/ITP30002] Operating Systems

Mission

- Solve problem 1 and 2
  - Solve the problems on Ubuntu Linux (VirtualBox) using vim and gcc.
- Submission
  - Submit a .tgz file containing hw7_1.c and hw7_2.c on LMS.
    "tar cvfz hw7_<*student_id*>.tgz hw7_1.c hw7_2.c"
    (Do not copy&past but type the above command)
  - After compression, please check the .tgz file by decompressing in an empty directory.
    "tar xvfz hw7_<*student_id*>.tgz"
- Due date: PM 11:00:00, June 4th

Honor Code Guidelines

- “과제”
  - 과제는 교과과정의 내용을 소화하여 실질적인 활용 능력을 갖추기 위한 교육활동이다. 학생은 모든 과제를 정직하고 성실하게 수행함으로써 과제에 의도된 지식과 기술을 얻기 위해 최선을 다해야 한다.
  - 제출된 과제물은 성적 평가에 반영되므로 공식적으로 허용되지 않은 자료나 도움을 획득, 활용, 요구, 제공하는 것을 포함하여 평가의 공정성에 영향을 미치는 모든 형태의 부정행위는 단호히 거부해야 한다.
  - 수업 내용, 공지된 지식 및 정보, 또는 과제의 요구를 이해하기 위하여 동료의 도움을 받는 것은 부정행위에 포함되지 않는다. 그러나, 과제를 해결하기 위한 모든 과정은 반드시 스스로의 힘으로 수행해야 한다.
  - 담당교수가 명시적으로 허락한 경우를 제외하고 다른 사람이 작성하였거나 인터넷 등에서 획득한 과제물, 또는 프로그램 코드의 일부, 또는 전체를 이용하는 것은 부정행위에 해당한다.
  - 자신의 과제물을 타인에게 보여주거나 빌려주는 것은 공정한 평가를 방해하고, 해당 학생의 학업 성취를 저해하는 부정행위에 해당한다.
  - 팀 과제가 아닌 경우 두 명 이상이 함께 과제를 수행하여 이를 개별적으로 제출하는 것은 부정행위에 해당한다.
  - 스스로 많은 노력을 한 후에도 버그나 문제점을 파악하지 못하여 동료의 도움을 받는 경우도 단순한 문법적 오류에 그쳐야 한다. 과제가 요구하는 design, logic, algorithm의 작성에 있어서 담당교수, TA, tutor 이외에 다른 사람의 도움을 받는 것은 부정행위에 해당한다.
  - 서로 다른 학생이 제출한 제출물 간 유사도가 통상적으로 발생할 수 있는 정도를 크게 넘어서는 경우, 또는 자신이 제출한 과제물에 대하여 구체적인 설명을 하지 못하는 경우에는 부정행위로 의심받거나 판정될 수 있다.

Problem 0

- Search Internet to learn how to use following system calls and data type. #include <pthread.h>
  - POSIX mutex lock functions
    - pthread_mutex_init()
    - pthread_mutex_lock()
    - pthread_mutex_trylock()
    - pthread_mutex_unlock()
    - pthread_mutex_destroy()

    #include <semaphore.h>
  - POSIX semaphore functions
    - sem_init()
    - sem_wait()
    - sem_post()
    - sem_getvalue()
    - sem_destroy()

int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutex_attr *attr)
mutex는 thread가 공유하는 data 영역을 보호하기 위해서 사용
mutex 객체를 초기화 시키기 위해서 사용한다.
성공시 : return 0
실패시 : return -1

int pthread_mutex_lock(pthread_mutex_t *mutex)
critical section에 진입하기위해 mutex lock 요청
if mutex의 초기 상태가 unlocked라면 thread는 lock이되고 critical section에 들어가고 return한다.
다른 thread가 mutex lock 상태라면 lock을 얻을 수 있을 때까지 기다리게 된다.
성공시 : return 0
실패시 : return 0이 아닌 error 값

int pthread_mutex_trylock(pthread_mutex_t *mutex)
pthread_mutex_lock과 동일하게 동작하지만 mutex가 다른 thread에 의해
이미 lock된 상태에 있는 경우 즉시 return, mutex type이 pthread_mutex_recursive이고
mutex를 현재 호출 thread가 소유하고 있으면 mutex 카운트가 +1이되고 pthread_mutex_trylock 함수가
성공을 return
성공시 : return 0
실패시 : return 0이 아닌 error 값

int pthread_mutex_unlock(pthread_mutex_t *mutex)
mutex lock을 되돌려 준다. critical section에서 나올 때
mutex type이 pthread_mutex_recursive이고 mutex를 현재 호출 thread가
소유하고 있으면 mutex 카운트가 -1이되고 pthread_mutex_trylock 함수가 성공을 return
성공시 : return 0
실패시 : return 0이 아닌 error 값

int pthread_mutex_destory(pthread_mutex_t *mutex)
mutex가 가리키는 mutex 객체를 파기한다.
해당 mutex 객체가 더이 되지 X 상태로 만드는 효과가 있다
성공시 : return 0
실패시 : return 0이 아닌 error 값

int sem_init(sem_t *sem, int pshared, unsigned int value)

이름 없는 Semaphore 초기화 < 성공시 : return 0
실패시 : return -1

int sem_wait(sem_t *sem)

sem 값이 1 이상이면 감소시키고 함수종료 < 성공시 : return 0
실패시 : return -1

≈ lock

int sem_post sem_t *sem)

sem 값이 1 이상이면 증가 시키고 함수종료 < 성공시 : return 0
실패시 : return -1

≈ unlock

int sem_getvalue(sem_t *sem, int *sval)

sval 이 가리키는 위치에 sem Semaphore의 현재값 저장 < 성공시 : return 0
실패시 : return -1

int sem_destory(sem_t *sem)

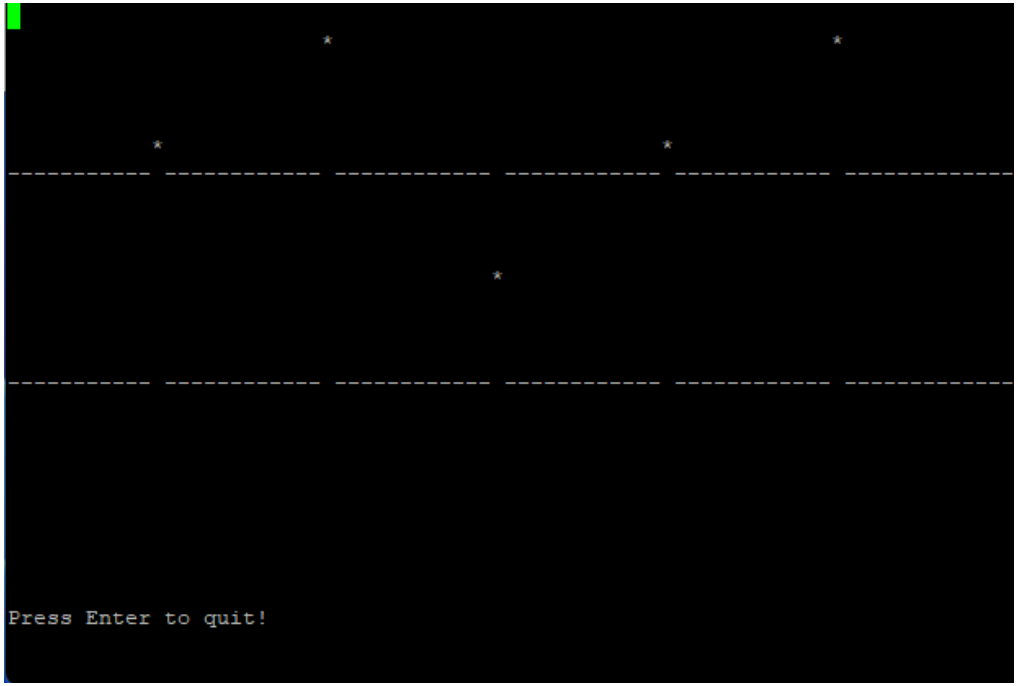
Semaphore 객체를 삭제하고 할당된 자원을 해제
sem_destory 를 호출하는 시점에서 해당 Semaphore를 기다리는 thread가 없어야 한다. < 성공시 : return 0
실패시 : return -1

Problem 0

- Read Console.h, Console.c and hw7_1_skeleton.c carefully to learning the usage of the following functions:
  - getWindowWidth(), getWindoHeight()
  - clrscr(), gotoxy(),
  - PrintXY(), DrawLine()

Problem 1

- Write a program that moves balls using multiple threads.
  - Complete the skeleton code so that only one ball can enter in the critical region at a time. (use pthread_mutex_t)

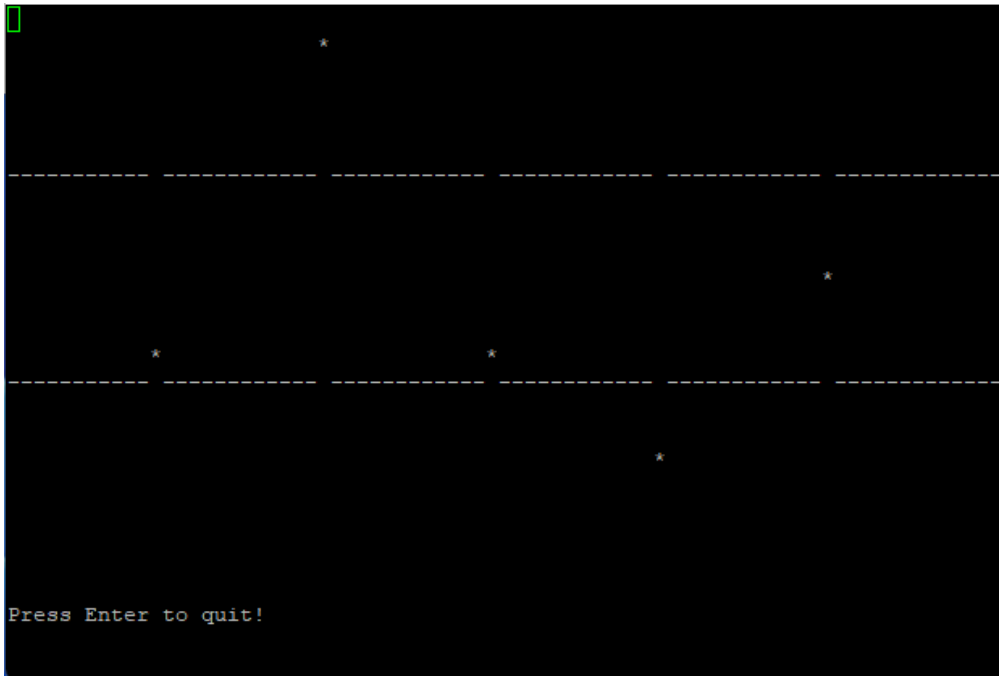


Problem 1

- The program should run similar to hw7_1.mp4.
- Follow the instructions in the skeleton code.
  - cp hw7_skeleton.c hw7_1.c
  - First, read the skeleton code carefully and understand all the techniques demonstrated in the code.
- Compilation
  - gcc hw7_1.c Console.c –pthread –o hw7_1
- Execution
  - ./hw7_1 <no_thread>

Problem 2

- Write a program that moves balls using multiple threads.
  - Complete the skeleton code so that only a specified number of balls can enter in the critical region at a time. (use sem_t)



Problem 2

- The program should run similar to hw7_2.mp4.
- Follow the instructions in the skeleton code.
  - cp hw7_skeleton.c hw7_2.c
  - First, read the skeleton code carefully and understand all the techniques demonstrated in the code.
- Compilation
  - gcc hw7_2.c Console.c –pthread –o hw7_2
- Execution
  - ./hw7_2 <no_thread> <no_ticket>